Backus–Naur Form

From Wikipedia, the free encyclopedia

In computer science, **Backus–Naur Form** (**BNF**) is a metasyntax used to express context-free grammars: that is, a formal way to describe formal languages. John Backus and Peter Naur developed a context free grammar to define the syntax of a programming language by using two sets of rules: i.e., lexical rules and syntactic rules.

BNF is widely used as a notation for the grammars of computer programming languages, instruction sets and communication protocols, as well as a notation for representing parts of natural language grammars. Many textbooks for programming language theory and/or semantics document the programming language in BNF.

There are many extensions and variants of BNF, including *Extended* and *Augmented* Backus–Naur Forms (EBNF and ABNF).

Contents

- 1 History
- 2 Introduction
- 3 Example
- 4 Further examples
- 5 Variants
- 6 See also
  - 6.1 Software using BNF
- 7 References
- 8 External links
  - 8.1 Language Grammars

History

John Backus created the notation in order to express the grammar of ALGOL. At the first World Computer Congress, which took place in Paris in 1959, Backus presented "The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM Conference", a formal description of the international algebraic language(IAL) which was later called ALGOL 58. The formal language he presented was based on Emil Post's production system. Generative grammars were an active subject of mathematical study, e.g. by Noam Chomsky, who was applying them to the grammar of natural language.[1][2]

Peter Naur (ALGOL 60, 1963) identified Backus's notation as **Backus Normal Form**, and simplified it to minimize the character set used, and, at the suggestion of Donald Knuth, his name was added in recognition of his contribution, his initial replacing the N for "Normal" since, Knuth argued, the BNF is "not a normal form in any sense".[3] The Backus–Naur Form or BNF grammars have significant similarities to Pāṇini's grammar rules, and the notation is sometimes also referred to as **Panini–Backus Form.**[4]

Introduction

A BNF specification is a set of derivation rules, written as

```
<symbol> ::= __expression__
```

where <symbol> is a *nonterminal*, and the __expression__ consists of one or more sequences of symbols; more sequences are separated by the vertical bar, '|', indicating a choice, the whole being a possible substitution for the symbol on the left. Symbols that never appear on a left side are *terminals*. On the other hand, symbols that appear on a left side are *non-terminals* and are always enclosed between the pair <>.

Example

As an example, consider this possible BNF for a U.S. postal address:

```
<postal-address> ::= <name-part> <street-address> <zip-part>

     <name-part> ::= <personal-part> <last-name> <opt-jr-part> <EOL>
                   | <personal-part> <name-part>

 <personal-part> ::= <first-name> | <initial> "."

<street-address> ::= <house-num> <street-name> <opt-apt-num> <EOL>

      <zip-part> ::= <town-name> "," <state-code> <ZIP-code> <EOL>

   <opt-jr-part> ::= "Sr." | "Jr." | <roman-numeral> | ""
```

This translates into English as:

- A postal address consists of a name-part, followed by a street-address part, followed by a zip-code part.
- A name-part consists of either: a personal-part followed by a last name followed by an optional "jr-part" (Jr., Sr., or dynastic number) and end-of-line, or a personal part followed by a name part (this rule illustrates the use of recursion in BNFs, covering the case of people who use multiple first and middle names and/or initials).
- A personal-part consists of either a first name or an initial followed by a dot.
- A street address consists of a house number, followed by a street name, followed by an optional apartment specifier, followed by an end-of-line.
- A zip-part consists of a town-name, followed by a comma, followed by a state code, followed by a ZIP-code followed by an end-of-line.

Note that many things (such as the format of a first-name, apartment specifier, ZIP-code, and Roman numeral) are left unspecified here. If necessary, they may be described using additional BNF rules.

Further examples

BNF's syntax itself may be represented with a BNF like the following:

```
<syntax> ::= <rule> | <rule> <syntax>
<rule>   ::= <opt-whitespace> "<" <rule-name> ">" <opt-whitespace> "::="
                 <opt-whitespace> <expression> <line-end>
<opt-whitespace> ::= " " <opt-whitespace> | ""  <!-- "" is empty string, i.e. no whit
<expression>     ::= <list> | <list> "|" <expression>
<line-end>       ::= <opt-whitespace> <EOL> | <line-end> <line-end>
<list>    ::= <term> | <term> <opt-whitespace> <list>
<term>    ::= <literal> | "<" <rule-name> ">"
<literal> ::= '"' <text> '"' | "'" <text> "'" <!-- actually, the original BNF did not
```

This assumes that no whitespace is necessary for proper interpretation of the rule. <EOL> represents the appropriate line-end specifier (in ASCII, carriage-return and/or line-feed, depending on the operating system). <rule-name> and <text> are to be substituted with a declared rule's name/label or literal text, respectively.

Variants

There are many variants and extensions of BNF, generally either for the sake of simplicity and succinctness, or to adapt it to a specific application. One common feature of many variants is the use of regular expression repetition operators such as * and +. The Extended Backus–Naur Form (EBNF) is a common one. In fact the example above is not the pure form invented for the ALGOL 60 report. The bracket notation "[]" was introduced a few years later in IBM's PL/I definition but is now universally recognised. ABNF and RBNF are other extensions commonly used to describe IETF protocols.

Parsing expression grammars build on the BNF and regular expression notations to form an alternative class of formal grammar, which is essentially analytic rather than generative in character.

Many BNF specifications found online today are intended to be human readable and are non-formal. These often include many of the following syntax rules and extensions:

- Optional items enclosed in square brackets. E.g. [<item-x>]
- Items repeating 0 or more times are enclosed in curly brackets or suffixed with an asterisk. E.g. <word> ::= <letter> {<letter>}
- Items repeating 1 or more times are followed by a '+'
- Terminals may appear in bold and NonTerminals in plain text rather than using italics and angle brackets
- Alternative choices in a production are separated by the '|' symbol. E.g., <alternative-A> | <alternative-B>
- Where items need to be grouped they are enclosed in simple parentheses

See also

- Extended Backus–Naur Form.
- Ashtadhyayi (Sanskrit grammar with mathematical structure).
- Syntax diagram (Railroad diagram).
- Definite clause grammar A more expressive alternative to BNF used in Prolog
- Wirth syntax notation An alternative to BNF from 1977.
- John Backus

Software using BNF

- ANTLR Another parser generator written in Java.
- BNFC (http://www.cs.chalmers.se/Cs/Research/Language-technology/BNFC/) BNF Converter.
- GOLD BNF parser.
- GNU bison GNU version of yacc.
- Yacc parser generator (used with Lex pre-processor).

References

This article was originally based on material from the Free On-line Dictionary of Computing, which is licensed under the GFDL.

1. ^ Chomsky, Noam (1956). "Three Models for the Description of Language". *IRE Transactions on Information Theory* **Vol. 2** (No. 2): 113–123. doi:10.1109/TIT.1956.1056813.

2. ^ Chomsky, Noam (1957). *Syntactic Structures*. The Hague: Mouton.
3. ^ Knuth, Donald E. (1964). "Backus Normal Form vs. Backus Naur Form". *Communications of the ACM* **7** (12): 735–736. doi:10.1145/355588.365140.
4. ^ P.Z. Ingerman (1967)

External links

- Backus Normal Form vs. Backus-Naur Form (http://compilers.iecc.com/comparch/article/93-07-017) explains some of the history of the two names.
- BNF and EBNF: What are they and how do they work? (http://www.garshol.priv.no/download/text/bnf.html) by Lars Marius Garshol.
- RFC 4234 (ftp://ftp.rfc-editor.org/in-notes/rfc4234.txt) Augmented BNF for Syntax Specifications: ABNF.
- Reduced BNF (RBNF) A Syntax Used in Various Protocol Specifications (http://tools.ietf.org/html/draft-farrel-rtg-common-bnf) Draft.
- Comparision of different variants of BNF (http://www-cgi.uni-regensburg.de/~brf09510/grammartypes.html)
- Syntax diagram of EBNF (http://www-cgi.uni-regensburg.de/~brf09510/syntax/lazyebnf.ebnf.html)
- Generation of syntax diagrams from EBNF (http://www-cgi.uni-regensburg.de/~brf09510/syntax.html)
- ISO/IEC 14977:1996(E) *Information technology - Syntactic metalanguage - Extended BNF*, available from ISO (http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html) or from Marcus Kuhn (http://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf) (the latter is missing the cover page, but is otherwise much cleaner)

Language Grammars

- Algol-60 BNF (http://www.lrz-muenchen.de/~bernhard/Algol-BNF.html) , the original BNF.
- BNF grammars for SQL-92, SQL-99 and SQL-2003 (http://savage.net.au/SQL/index.html) , Freely available BNF grammars for SQL.
- BNF Web Club (http://cui.unige.ch/db-research/Enseignement/analyseinfo/BNFweb.html) , Freely available BNF grammars for SQL, Ada, Java.
- BNF Examples and Full Programming Languages (http://www.devincook.com/GOLDParser/grammars/index.htm) , Freely available BNF grammars for programming languages (C, Java, JavaScript, C#, VB.NET, SQL-89), Academic Languages, File Formats (HTML, XML, CSS) and others (Regular Expressions, YACC). These are written in the GOLD Meta-Language (http://www.devincook.com/GOLDParser/doc/meta-language/index.htm) , consisting of BNF and Regular Expressions.
- Free Programming Language Grammars for Compiler Construction (http://www.thefreecountry.com/sourcecode/grammars.shtml) , Freely available BNF/EBNF grammars for C/C++, Pascal, COBOL, Ada 95, PL/I.

Retrieved from "http://en.wikipedia.org/wiki/Backus%E2%80%93Naur_Form"
Categories: Formal languages | Compiler theory
Hidden categories: Wikipedia articles incorporating text from FOLDOC